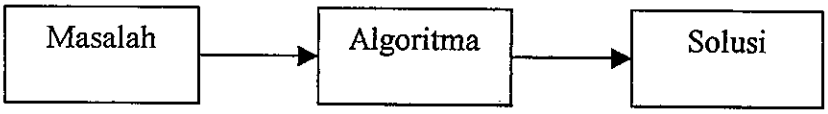
BAB II

TEORI PENUNJANG

2.1. Dasar – dasar Algoritma

Algoritma didefinisikan sebagai suatu metode yang terdiri dari serangkaian langkah terstruktur dan ditulis secara sistematis yang akan dikerjakan dalam menyelesaikan suatu masalah untuk mendapatkan solusi dengan bantuan komputer.

Hubungan antara masalah, algoritma, dan solusi dapat digambarkan sebagai berikut :



Gambar 2.1. Hubungan masalah, algoritma dan solusi

Proses dari masalah hingga terbentuk suatu algoritma disebut tahap pemecahan masalah. Proses dari algoritma hingga terbentuk suatu solusi disebut dengan tahap implementasi.

Algoritma pemrograman yang baik memiliki ciri – ciri sebagai berikut :

1. Memiliki metode yang tepat dalam memecahkan masalah.
2. Menghasilkan keluaran yang benar.
3. Ditulis secara sistemetis, sehingga dapat dipahami dan tidak menimbulkan arti ganda.
4. Operasi yang dibutuhkan terdefinisi dengan jelas.
5. Proses harus berakhir setelah sejumlah langkah dilakukan.

2.2. Pengurutan (Sorting)

Pengurutan data (*sorting*) secara umum dapat didefinisikan sebagai suatu proses untuk menyusun kembali himpunan objek (data) menggunakan aturan tertentu. Ada dua jenis pengurutan data, yaitu pengurutan secara urut naik (*ascending*), dan pengurutan secara urut turun (*descending*).

Keuntungan dari data yang sudah dalam keadaan urut antara lain bahwa data lebih mudah untuk diolah kembali, misalnya dicari, dihapus, disisipi maupun digabungkan.

Data yang akan diurutkan tentunya bervariasi baik dalam hal banyak data maupun jenisnya, tetapi tidak ada algoritma yang terbaik untuk setiap situasi yang dihadapi. Bahkan tidak dapat ditentukan algoritma yang paling baik untuk situasi tertentu.

Ada beberapa faktor yang mempengaruhi efektifitas pengurutan data. Diantaranya adalah banyak dan jenis data, kapasitas pengingat, serta tempat penyimpanan data. Hal tersebut tentunya berpengaruh terhadap waktu yang diperlukan untuk pengurutan data. Oleh karena itu pemilihan algoritma sangat penting untuk menunjang efektifitas pengurutan data.

Metode pengurutan dapat diklasifikasikan menjadi 2 kategori, yaitu :

1. Pengurutan internal.

   Pengurutan internal merupakan metode pengurutan yang dilakukan di dalam memori utama komputer. Dalam pengurutan internal, metode yang digunakan harus ekonomis dalam penggunaan media penyimpan yang tersedia. Yang termasuk dalam pengurutan internal adalah pengurutan

larik, karena larik tersimpan dalam memori utama komputer.. Metode yang digunakan dalam pengurutan internal adalah :

- Sorting by Insertion (penyisipan).

  Prinsip dari metode ini adalah membandingkan data (x_i) dengan data sebelumnya, kemudian menyisipkan data (x_i) ke posisi yang tepat. Proses ini berlanjut sampai semua data terurutkan. Yang termasuk dalam metode ini adalah Straight Insertion Sort, Binary Insertion Sort dan Shell Sort.

- Sorting by Selection (seleksi).

  Prinsip dari metode ini adalah mencari data terkecil dari data pertama sampai terakhir, kemudian data terkecil tersebut ditukar dengan data pertama. Selanjutnya mencari data terkecil dari data kedua sampai data terakhir, kemudian data terkecil tersebut ditukar dengan data kedua. Proses ini berlanjut sampai semua data terurutkan. Yang termasuk dalam metode ini adalah Straight Selection Sort.

- Sorting by Exchange (penukaran).

  Prinsip dari metode ini berdasarkan pada konsep pembandingan dan penukaran sepasang data sampai semua data terurutkan. Yang termasuk dalam metode ini adalah Bubble Sort, Shaker Sort dan Quick Sort.

  Ketiga metode di atas disebut juga dengan metode langsung, karena proses penyisipan, seleksi atau penukaran dilakukan di dalam 1 larik.

- Sorting by Merging (penggabungan).

  Prinsip dari metode ini adalah membandingkan dan menggabungkan data dari dua buah vektor yang masing - masing telah terurutkan. Yang termasuk dalam metode ini adalah Merge Sort.

- Sorting by Distibution (distribusi).

  Prinsip dari metode ini menggunakan 2 buah larik. Larik pertama berisi data-data yang akan diurutkan dan larik kedua merupakan larik yang disediakan untuk pendistribusian data dari larik pertama. Selanjutnya data-data dari larik pertama didistribusikan ke dalam larik kedua sesuai dengan tempat yang telah disediakan. Yang termasuk dalam metode ini adalah Radix Sort dan Postman Sort.

2. Pengurutan eksternal.

   Pengurutan eksternal merupakan metode pengurutan yang datanya tersimpan dalam pangingat luar, misalnya cakram atau pita magnetis. Yang termasuk dalam pengurutan eksternal adalah pengurutan berkas. Prinsip dari pengurutan eksternal ini sama dengan Merge Sort. Data dengan jumlah N rekaman pada berkas dipecah menjadi x bagian (berkas), kemudian data pada masing-masing berkas diurutkan. Selanjutnya setelah x berkas diurutkan secara terepisah, dilakukan merging terhadap x berkas tersebut untuk mendapatkan berkas yang terdiri dari N rekaman yang sudah terurutkan.

2.3. Bahasa Pemrograman Pascal

Menurut jenisnya bahasa pemrograman pada komputer dibagi menjadi 2 kelompok, yaitu :

- Bahasa pemrograman tingkat rendah (*low level programming language*).
- Bahasa pemrograman tingkat tinggi (*high level programming language*).

Bahasa pemrograman tingkat rendah bersifat *machine dependent*, yang artinya bahwa bahasa tingkat rendah untuk suatu komputer akan berbeda dengan komputer lain. Yang termasuk bahasa pemrograman tingkat rendah adalah bahasa mesin.

Bahasa pemrograman tingkat tinggi adalah bahasa pemrograman yang dapat dipahami dan tidak berpengaruh pada komputer yang digunakan. Saat ini banyak dijumpai bahasa pemrograman tingkat tinggi dengan berbagai macam versi dan fasilitas yang disediakan, diantaranya adalah Pascal, BASIC, FORTRAN, C dan COBOL.

Bahasa pemrograman Pascal merupakan bahasa pemrograman dengan teknik pemrograman terstruktur. Pascal diambil dari nama seorang sarjana Perancis, Blaise Pascal. Pertama kali dikembangkan oleh Niklaus Wirth, seorang ahli ilmu komputer dari Swiss, pada tahun 1970.

Secara umum struktur program Pascal adalah sebagai berikut :

```
program BAGAN_PROGRAM;        {*** nama program ***}
uses .....          {*** deklarasi piranti ***}
label .....         {*** deklarasi label ***}
```

```
const .....      {*** deklarasi konstanta ***}
type .....       {*** deklarasi tipe data ***}
var .....        {*** deklarasi peubah ***}
procedure SATU;
procedure DUA;
function TAMBAH;
begin            {*** awal program utama ***}
.....
.....
< pernyataan – pernyataan dari program BAGAN_PROGRAM >
.....
.....
end.             {*** akhir program utama ***}
```

Gambar 2.2. Struktur program Pascal

2.3.1. Pernyataan Perulangan

Pernyataan perulangan (*repetitive pernyataan*) digunakan untuk melakukan proses berulang terhadap pernyataan tunggal maupun pernyataan majemuk. Yang termasuk pernyataan perulangan adalah pernyataan **for**, pernyataan **repeat...until** dan pernyataan **while.**

2.3.1.1. Pernyataan for

Proses berulang dalam pernyataan **for** langsung dikendalikan oleh suatu peubah yang disebut dengan peubah kendali (*control variable*).

Bentuk umum pernyataan **for** :

For *peubah* := *awal* **to** *akhir* **do** *pernyataan*;

dengan peubah : nama peubah kendali.

awal : nilai awal peubah kendali.

akhir : nilai akhir peubah kendali.

pernyataan : pernyataan yang akan diproses berulang.

Bentuk umum diatas dapat dijelaskan sebagai berikut. Pertama kali nilai *peubah* dibuat sama dengan *awal* dan *pernyataan* akan dikerjakan. Selanjutnya nilai *peubah* ditambah dengan 1, dan *pernyataan* dikerjakan lagi. Proses ini diulang sampai nilai *peubah* lebih dari *akhir*. Jika nilai awal lebih besar dari nilai *akhir*, maka *pernyataan* tidak dikerjakan.

Contoh pemakaian pernyataan **for** :

```
for i := 1 to 3 do
    writeln('Test', i);
```

Jika potongan program ini dijalankan, maka akan dihasilkan keluaran :

Test 1

Test 2

Test 3

Penulisan pernyataan **for** dapat divariasi menjadi :

for *peubah* := *awal* **downto** *akhir* **do** *pernyataan*;

Kata **downto** menunjukkan bahwa setelah *pernyataan* dikerjakan satu kali, maka nilai *peubah* dikurangi dengan 1, sampai akhirnya nilai *peubah* kurang dari nilai *akhir*. Dengan demikian kata **downto** harus digunakan jika nilai *awal* lebih besar dari nilai *akhir*.

Contoh :

```
for i := 3 downto 1 do
    writeln('Test', i);
```

Jika potongan program ini dijalankan, maka akan dihasilkan keluaran :

Test 3

Test 2

Test 1

2.3.1.2. Pernyataan repeat...until

Pernyataan **repeat** dipakai sebagai awal pernyataan dan **until** dipakai sebagai akhir pernyataan yang dikerjakan berulang sekaligus untuk mencek apakah proses berulang tersebut masih bisa diteruskan atau sudah harus berakhir.

Bentuk umum pernyataan **repeat...until** :

repeat *pernyataan* **until** *kondisi*;

dengan pernyataan : pernyataan yang akan dikerjakan berulang.

kondisi : syarat agar proses berulang dihentikan.

Contoh pemakaian pernyataan **repeat...until** :

```
mulai := 1;
repeat
    writeln('Test');
    mulai := mulai + 1
until mulai > 5;
```

Jika potongan program tersebut dijalankan, akan dihasilkan / dicetak kata **Test** sebanyak 5 kali.

2.3.1.3. Pernyataan while

Pernyataan **while** hampir sama dengan **repeat...until** dengan perbedaan bahwa dalam pernyataan **while** kondisi untuk melakukan proses berulang ditest di awal, sedangkan pada pernyataan **repeat...until** di bagian akhir.

Bentuk umum pernyataan **while** :

while *kondisi* **do** *pernyataan*;

dengan kondisi : syarat agar proses berulang berjalan.

pernyataan : pernyataan yang akan diproses berulang.

Contoh pemakaian pernyataan **while** :

```
mulai := 5;
while mulai > 0 do
    begin
    writeln('Test');
    mulai := mulai - 1;
    end;
```

Jika potongan program tersebut dijalankan, akan dihasilkan / dicetak kata **Test** sebanyak 5 kali.

2.3.2. Pernyataan Penyeleksian kondisi

Pernyataan penyeleksian kondisi digunakan untuk memilih bagian program yang akan dikerjakan sesuai dengan kondisi yang diberikan. Ada 2 buah pernyataan penyeleksian kondisi, yaitu pernyataan **if** dan pernyataan **case**.

2.3.2.1. Pernyataan if

Pernyataan **if** digunakan untuk mencek suatu kondisi dan menentukan apakah kondisi tersebut benar atau salah, kemudian melakukan suatu kegiatan sesuai dengan nilai kondisi tersebut.

Bentuk umum pernyataan **if** :

if *kondisi* **then** *pernyataan1* < **else** *pernyataan2* >;

dengan kondisi : kondisi yang ditest untuk menentukan apakah *pernyataan1* atau *pernyataan2* yang akan dikerjakan.

pernyataan1 : pernyataan yang akan dikerjakan jika *kondisi* bernilai benar.

pernyataan2 : pernyataan yang akan dikerjakan jika *kondisi* bernilai salah.

Contoh pemakaian pernyataan **if** :

```
if x = 100 then
   x := x /5
```

else

x := x –1;

2.3.2.2. Pernyataan case

Pernyataan **case** berisi ungkapan dan sederetan pernyataan yang masing–masing diawali dengan satu atau lebih konstanta atau dengan kata **else**.

Bentuk umum pernyataan **case** :

case *pemilih* **of**

Konstanta1 : pernyataan1;

Konstanta2 : pernyataan2;

.....

.....

< **else** : *pernyataann* ;>

end;

dengan pemilih : nama peubah sebagai pemilih;

konstanta1, konstanta2, ... : kemungkinan–kemungkinan nilai pemilih.

Pernyataan1, pernyataan2, ... : pernyataan yang akan dikerjakan sesuai dengan nilai pemilih.

Contoh pemakaian pernyataan **case** :

case *bilangan* **of**

1..10 : **writeln**('Antara 1 sampai 10');

11..100 : **writeln**('Antara 11 sampai 100');

```
        else    : writeln('Lebih besar dari 100')
    end;
```

2.3.3. Prosedur dan Fungsi

Prosedur dan fungsi merupakan sekelompok pernyataan yang seolah-olah terpisah dari program utama tetapi sesungguhnya merupakan bagian atau sub bagian dari program utama. Prosedur diaktifkan menggunakan pernyataan **procedure** (pemanggil prosedur), dan fungsi diaktifkan dengan suatu ungkapan yang hasilnya akan dikembalikan lagi sebagai nilai baru dari ungkapan tersebut (nama fungsi sekaligus berfungsi sebagai nama peubah).

Contoh prosedur untuk menghitung luas suatu persegi dengan sisi-sisi x dan y :

```
procedure LUAS;
var x, y, L : real;
begin
    L := x * y;
end;
```

Contoh fungsi untuk menghitung luas suatu persegi dengan sisi-sisi x dan y :

```
function LUAS;
var x, y, Luas : real;
begin
    Luas := x * y;
end;
```

2.3.3.1. Prosedur

Prosedur mempunyai struktur yang sama dengan struktur program, yaitu terdiri dari nama prosedur, deklarasi-deklarasi dan bagian utama dari prosedur itu sendiri. Deklarasi dalam prosedur disebut sebagai deklarasi lokal, sehingga hanya bisa digunakan dalam prosedur itu saja dan tidak dikenal di luar prosedur.

Bentuk umum deklarasi prosedur :

procedure *nama* <(*dafpar*)>;

dengan nama : nama prosedur.

dafpar : daftar parameter formal.

Contoh pemakaian deklarasi prosedur :

procedure BACA_MATRIX (Var Mat : Tabel; N : Integer);

2.3.3.2. Fungsi

Secara umum fungsi sama dengan prosedur, dengan perbedaan bahwa nama fungsi sekaligus berfungsi sebagai nama peubah, sehingga dalam deklarasi fungsi harus dinyatakan tipe datanya.

Bentuk umum deklarasi fungsi :

function *nama* <(*dafpar*)> : *tipe*;

dengan nama : nama fungsi.

dafpar : daftar parameter formal.

tipe : tipe data dari fungsi tersebut.

Dalam fungsi semua parameter formal harus berupa parameter nilai, tidak diperbolehkan ada parameter peubah.

Contoh pemakaian deklarasi fungsi untuk menghitung nilai rata-rata dari sekumpulan data yang diketahui :

```
function RATA ( Vektor : Larik; N : Integer);
var I : integer;
    R : real;
begin
      R := 0;
      for I := 1 to N do
            R := R + Vektor[I];
      RATA := R / N
end.
```

2.3.4. Larik (Array)

Larik (**array**) adalah tipe terstruktur yang mempunyai komponen dalam jumlah yang tetap dan setiap komponen mempunyai tipe data yang sama. Posisi masing-masing komponen dalam larik dinyatakan sebagai nomor index.

Bentuk umum dari deklarasi tipe larik :

Type *pengenal* = **array** [*tipe_index*] of *tipe*;

dengan pengenal : nama tipe data.

tipe_index : tipe data untuk nomor index.

tipe : tipe data komponen.

Parameter *tipe_index* menentukan banyaknya komponen dalam larik tersebut. Parameter ini adalah sembarang tipe ordinal kecuali **longint** dan subjangkauan dari **longint.**

Contoh pemakaian deklarasi tipe larik :

type Nilai = **array** [1..100] **of integer**;

Contoh ini menunjukkan bahwa *nilai* adalah tipe data yang berupa larik yang komponennya bertipe integer dan banyaknya 100 buah.

2.4. Kompleksitas algoritma

Salah satu alat yang digunakan untuk mengetahui efisiensi suatu algoritma adalah waktu yang diperlukan oleh komputer untuk menyelesaikan suatu permasalahan dengan menggunakan algoritma tersebut berdasarkan input yang diberikan.

Penyataan tentang hal tersebut di atas termasuk dalam kompleksitas perhitungan suatu algoritma. Kompleksitas waktu suatu algoritma adalah analisa waktu yang diinginkan untuk menyelesaikan suatu permasalahan. Analisa memory yang diinginkan termasuk dalam kompleksitas ruang suatu algoritma. Pertimbangan kompleksitas waktu dan ruang pada suatu algoritma merupakan suatu hal yang penting ketika suatu algoritma akan diimplementasikan.

2.5. Waktu Tempuh (Running Time)

Dengan diketahui kompleksitas waktu suatu algoritma maka efisiensi program berdasarkan waktu dapat dibedakan menurut *Running Time*-nya. Waktu

tempuh (*running time*) dari suatu algoritma adalah ukuran waktu untuk melaksanakan suatu program sehingga menghasilkan output pada suatu kompiler dan mesin eksekusi tertentu.

Dalam analisa algoritma, Running Time dinyatakan dalam simbol f(n). Running Time sebuah program tergantung beberapa faktor :

1. Input program.
2. Kualitas / kemampuan dari kompiler yang digunakan untuk kompilasi pada program.
3. Kemampuan dan kecepatan mesin yang digunakan untuk eksekusi program.
4. Kompleksitas waktu program.

Untuk menghitung waktu tempuh program, dikenal beberapa notasi diantaranya notasi O, notasi Θ, dan notasi Ω.

2.5.1. Notasi Big Oh

Definisi 2.1.

Jika f(n) adalah O(g(n)), maka terdapat dua buah konstanta bulat positif c dan m sedemikian hingga $f(n) \leq c.g(n)$, untuk $n \geq m$.

Penjelasan :

Untuk menunjukkan bahwa f(n) adalah O(g(n)), diperlukan dua buah konstanta c dan m sedemikian hingga $f(n) \leq c.g(n)$, untuk $n \geq m$. Konstanta c dan m yang memenuhi definisi tidak pernah tunggal. Jika konstanta c dan m ada, maka terdapat tidak terbatas konstanta c dan m yang lain, misalnya c_1 dan m_1, sedemikian hingga $c < c_1$ dan $m > m_1$ memenuhi definisi :

$f(n) \leq c.g(n) \leq c_1.g(n)$, untuk $n \geq m \geq m_1$.

Contoh :

Tunjukkan bahwa $f(n) = 5n + 8$ adalah $O(n)$.

Penyelesaian :

Menentukan dua buah konstanta c dan m sedemikian hingga $f(n) \leq c.g(n)$, untuk $n \geq m$. Jika $0 \leq 5n + 8 \leq 5n + 8n = 13n$, untuk $n \geq 1$, maka didapatkan konstanta $c = 13$ dan $m = 1$. Sehingga $f(n) \leq 13\ n$, untuk $n \geq 1$, dan terlihat bahwa f (n) adalah O(n).

Teorema 2.1.

$f(n) = a_d\ n^d + a_{d-1}\ n^{d-1} + \ldots. + a_1\ n + a_0$ adalah suatu fungsi polinomial dalam n dan derajat d, maka $f(n) = O(n^d)$.

Bukti :

Dengan merubah semua koefisien dari f(n) menjadi bilangan-bilangan positif maka nilai f(n) akan lebih besar untuk semua n bilangan integer positif. Jika ada $n^j \leq n^d$, untuk $j \leq d$, maka :

$$
\begin{aligned}
f(n) &= a_d\ n^d + a_{d-1}\ n^{d-1} + \ldots. + a_1\ n + a_0 \\
&= a_d\ n^d + a_{d-1}\ n^{d-1} + \ldots. + a_j\ n^j + \ldots. + a_0 \\
&\leq |a_d|\ n^d + \ldots. + |a_j|\ n^j + \ldots. + |a_0| \\
&\leq |a_d|\ n^d + \ldots. + |a_j|\ n^d + \ldots. + |a_0|\ n^d \\
&= (|a_d| + \ldots. + |a_j| + \ldots. + |a_0|)\ n^d \\
&= c.\ n^d
\end{aligned}
$$

dengan $c = (|a_d| + \ldots. + |a_j| + \ldots. + |a_0|)$, sehingga f(n) adalah $O(n^d)$.

Contoh :

Tunjukkan bahwa $f(n) = 3n^2 - 4n + 5$ adalah $O(n^2)$.

Penyelesaian :

Berdasarkan teorema 2.1, f(n) merupakan fungsi polinomial dalam n dengan derajat 2, maka f(n) adalah $O(n^2)$.

$$\begin{aligned} f(n) &= 3n^2 - 4n + 5 \\ &\leq 3n^2 + |-4|n + 5 \\ &\leq 3n^2 + |-4|n^2 + 5n^2 \\ &= (3+4+5)\, n^2 \\ &= 12\, n^2 \end{aligned}$$

Jadi terlihat bahwa $f(n) = 3n^2 - 4n + 5$ adalah $O(n^2)$.

Teorema 2.2.

Jika f(n) adalah O(g(n)) maka a.f(n) = O(g(n)).

Bukti :

Berdasarkan definisi 2.1, jika f(n) = O(g(n)) maka $f(n) \leq c.g(n)$. Terdapat konstanta c_1, sedemikian hingga $c_1 = \frac{c}{a}$ maka :

$$\begin{aligned} f(n) &\leq c_1 \,.\, g(n) \\ &\leq \frac{c}{a}\, g(n) \\ a \,.\, f(n) &\leq c \,.\, g(n) \end{aligned}$$

Contoh :

Tunjukkan bahwa $h(n) = 9n^2 - 12n + 15$ adalah $O(n^2)$.

Penyelesaian :

Dari contoh pada teorema 2.1, $f(n) = 3n^2 - 4n + 5$ adalah $O(n^2)$.

$h(n) = 9n^2 - 12n + 15 = 3(3n^2 - 4n + 5) = 3f(n)$. Maka menurut teorema 2.2, $3f(n)$ adalah $O(n^2)$.

Teorema 2.3.

Jika $f(n)$ adalah $O(g(n))$ dan $h(n)$ adalah $O(k(n))$, maka $f(n) + h(n) = O(g(n) + k(n))$.

Bukti :

$f(n)$ adalah $O(g(n))$ dan $h(n)$ adalah $O(k(n))$. Berdasar definisi 2.1, maka terdapat empat buah konstanta c_1 ,c_2 ,m_1 dan m_2, sehingga

$f(n) \leq c_1 . g(n)$, $n \geq m_1$

$h(n) \leq c_2 . k(n)$, $n \geq m_2$

$$\begin{aligned} f(n) + h(n) &= c_1 . g(n) + c_2 . g(n) \\ &\leq c . g(n) + c . g(n) \\ &= c\,(g(n) + k(n)) \\ &= O(g(n) + k(n)) \end{aligned}$$

Notasi $O(g(n) + k(n))$ dapat ditulis menjadi $O(\max(g(n) , k(n)))$, merupakan nilai maksimum dari $g(n)$ dan $k(n)$ yang disebut *maximum rule*.

Contoh :

$f(n) = 3n + 7$ adalah $O(n)$, dan $h(n) = 2n^2 - n + 8$ adalah $O(n^2)$. Tunjukkan bahwa $f(n) + h(n) = O(n^2)$.

Penyelesaian :

$f(n) + h(n) = 3n + 7 + 2n^2 - n + 8$

$= 2n^2 + 2n + 15$

$\leq 2n^2 + 2n^2 + 15n^2$

$= (2 + 2 + 15)n^2$

$= O(n^2)$

Teorema 2.4.

Jika f(n) adalah O(g(n)) dan h(n) adalah O(k(n)), maka $f(n) \cdot h(n) = O(g(n) \cdot k(n))$.

Bukti :

f(n) adalah O(g(n)) dan h(n) adalah O(k(n)). Berdasar definisi 2.1, maka terdapat empat buah konstanta c_1 ,c_2 ,m_1 dan m_2, sehingga

$f(n) \leq c_1 \cdot g(n)$, $n \geq m_1$

$h(n) \leq c_2 \cdot k(n)$, $n \geq m_2$

$f(n) \cdot h(n) = c_1 \cdot g(n) \cdot c_2 \cdot g(n)$

$\leq c \cdot g(n) \cdot c \cdot g(n)$

$= c\,(g(n) \cdot k(n))$

$= O(g(n) \cdot k(n))$

Contoh :

$f(n) = 2n^2 - 3$ adalah $O(n^2)$ dan $h(n) = n + 1$ adalah O(n). Tunjukkan bahwa $f(n) \cdot h(n) = O(n^3)$

Penyelesaian :

$$
\begin{aligned}
f(n) \cdot h(n) &= 2n^3 + 2n^2 - 3n - 3 \\
&\leq 2n^3 + 2n^2 + |-3|n + |-3| \\
&\leq 2n^3 + 2n^2 + 3n^3 + 3n^3 \\
&= (2+2+3+3)n^3 \\
&= 10\, n^3 \\
&= O(n^3)
\end{aligned}
$$

Dari definisi dan teorema-teorema yang telah dijelaskan, dapat dibuat suatu rumusan umum, yaitu $n^i = O(n^j)$, untuk $i \leq j$.

2.5.2. Analisa Algoritma

Jika ditentukan beberapa algoritma yang berbeda untuk menyelesaikan suatu permasalahan yang sama, maka harus dipilih salah satu yang sesuai dengan kebutuhan. Cara yang digunakan yaitu dengan analisa algoritma. Analisa algoritma dapat digunakan untuk menentukan efisiensi dari suatu algoritma.

Secara umum running time pada suatu pernyataan / kelompok pernyataan terparameterisasi oleh ukuran input / beberapa variabel. Parameter untuk running time suatu program adalah n (ukuran input). Adapun aturan umum untuk menganalisa suatu algoritma adalah :

1. Jika $f(n) = O(g(n))$ dan $h(n) = O(k(n))$, maka
   a. $f(n) + h(n) = O(\max(g(n), k(n))$.
   b. $f(n) \cdot h(n) = O(g(n) \cdot k(n))$.

2. Running Time suatu loop adalah running time dari besarnya jumlah iterasi pernyataan / kelompok pernyataan di dalam loop.
3. Total running time di dalam kelompok loop tersarang adalah running time dari pernyataan / kelompok pernyataan yang merupakan hasil kali dari besarnya keseluruhan loop.
4. Running Time setiap *assigment* (tugas), *read* (baca), dan *write* (tulis) adalah O(1).
5. Running Time pada barisan pernyataan ditentukan dengan aturan penjumlahan yang merupakan running time terbesar dari beberapa barisan pernyataan.
6. Running Time pada pernyataan *if* adalah harga pada kondisi pernyataan yang dieksekusi ditambah waktu untuk mengevaluasi kondisi. Waktu menghitung kondisi secara normal adalah O(1). Running time untuk *if..then..else* adalah waktu menghitung kondisi ditambah waktu terbesar yang dibutuhkan eksekusi pernyataan jika kondisi *false* (salah).
7. Waktu eksekusi adalah jumlah semua waktu sekitar loop, waktu eksekusi sekumpulan pernyataan dan waktu mengevaluasi kondisi untuk penghentian.

Metode yang digunakan untuk mencari running time dari suatu program adalah dengan membuat tabel dalam daftar jumlah total dari langkah-langkah yang telah dilaksanakan oleh setiap pernyataan. Notasi *cost* merupakan jumlah langkah dari suatu pernyataan setiap eksekusi. Sedangkan *times* adalah jumlah total waktu setiap pernyataan yang telah dieksekusi. Dengan mengkombinasikan keduanya, akan diperoleh perhitungan total setiap pernyataan. Selanjutnya dengan

penjumlahan total tiap pernyataan akan diperoleh running time dari suatu program.

Contoh :

Prosedur penjumlahan dua buah matrik A dan B ordo m x n, dengan hasil penjumlahan adalah matrik C.

```
procedure JUMLAH(var A, B, C : matrik; m, n : integer);
1. var i, j : integer;
2. begin
3.     for i := 1 to m do
4.         for j := 1 to n do
5.             C[i,j] := A[i,j] + B[i,j];
6. end;
```

Prosedur jumlah tersebut akan digunakan untuk mencari running time dari penjumlahan 2 buah matrik.

Line	Cost	Times	Total Steps
1	0	0	0
2	0	0	0
3	1	m + 1	m + 1
4	1	m (n + 1)	mn + m
5	1	mn	mn
6	0	1	0
Total number of steps			2mn + 2m + 1

Tabel 2.1. Contoh perhitungan kompleksitas waktu

Times pada baris ke 3 adalah m + 1 bukan m, hal ini karena i membutuhkan penambahan ke atas menjadi m + 1 sebelum loop diakhiri. Begitu juga *times* pada baris ke 4 yaitu m (n +1). Selanjutnya akan dicari analisa kompleksitas waktu untuk contoh program diatas.

Line	Cost	Times	Total Steps
1	0	0	O(0)
2	0	0	O(0)
3	1	m + 1	O(m)
4	1	mn + m	O(mn)
5	1	Mn	O(mn)
6	0	1	O(1)
f(m,n) = O(max(g(m,n) = O(mn)			

Tabel 2.2. Contoh analisa kompleksitas waktu

Berdasarkan teorema-teorema sub bab 2.5.1, penulisan tabel diatas dapat dijelaskan sebagai berikut :

- Pada line 1, 2 dan 6 mempunyai *total steps* 0, sehingga masing-masing mempunyai kompleksitas waktu O(0).
- Pada line 3 mempunyai *total steps* m + 1, sehingga

  $f(m) = m + 1$

  $0 \leq m + 1 \leq m + m = 2m$

  $f(m) \leq 2m$, maka $f(m) = O(m)$.

- Pada line 4 mempunyai *total steps* mn + m, sehingga

  $f(m,n) = mn + m$

  $0 \leq mn + m \leq mn + mn = 2mn$

  $f(m,n) \leq 2mn$, maka $f(m,n) = O(mn)$.

- Pada line 5 mempunyai *total steps* mn, sehingga

  $f(m,n) = mn$, maka $f(m,n) = O(mn)$.

  Sehingga $f(m,n) = 2mn + 2m + 1$

  $0 \leq 2mn + 2m + 1 \leq 2mn + 2mn + mn = 5mn$

  $f(m,n) \leq 5mn$, maka $f(m,n) = O(mn)$.